Cross-Platform Application Sharing Mechanism

Raja Naeem Akram, Konstantinos Markantonakis, and Keith Mayes
Information Security Group, Smart card Centre, Royal Holloway, University of London.
Egham, Surrey, United Kingdom.
Email: {RajaNaeem.Akram.2008, K.Markantonakis, Keith.Mayes}@rhul.ac.uk

***Abstract*—The application sharing mechanism in multi-application smart cards facilitates corroborative schemes between applications in a secure and reliable manner. Traditional application sharing can only be realised if both applications are installed on the same device. In this paper, we extend the smart card firewall to include the application sharing mechanism between applications installed on different smart cards. We propose Platform and Application Binding Protocols that enables two smart-cards / applications to authenticate and ascertain the trustworthiness before sharing resources. Furthermore, we provide an informal analysis of the protocols along with comparison with existing protocols. Subsequently, mechanical formal analysis based on the CasperFDR, and the implementation experience is presented.**

I. Introduction

The multi-application smart card initiative enabled a single device to host multiple applications in a secure and reliable manner. The smart card based firewall mechanisms provide a secure and reliable framework for resource sharing among corroborative applications, prohibiting unauthorised application communication. In traditional smart card architectures, a firewall only allows application sharing within the bounds of a smart card on which they are installed. Examples are Multos [1] and Java Card [2] firewall.

The Near Field Communication (NFC) [3] has reinvigorated the drive towards the multi–application smart card initiative. A user can have an NFC enabled smart phone that might have several applications on a secure element (e.g. smart card, embedded secure element, and secure memory card), which is a tamper-resistant device that can securely store and execute programs [4]. This also enables a user to have multiple secure elements in a mobile phone, where each can have a number of applications [5].

Furthermore, the User Centric Smart Card Ownership Model (UCOM) [6] gives the ownership of smart cards to their cardholders. The ownership entitles a user the "freedom of choice", which means that a user can only select an application for installation or deletion given that she is entitled to it. The application remains in total control of the application provider (or Service Provider: SP). The security and reliability assurance is provided by the platform itself; thereby, an SP does not have to trust the cardholder [7].

Therefore, there is a real possibility of having two applications that are in a corroborative scheme installed on different smart cards. The firewall mechanism of the UCOM [8] already provides an extension to the traditional mechanisms deployed in Multos and Java Card. In this paper, we further extend the architecture of the UCOM firewall [8] to accommodate the application sharing among applications that are installed on different smart cards. This extension to the UCOM firewall is referred as the Cross-Platform Application Sharing Mechanism (CPAM) and is the focus of this paper. To enable CPAM, two fundamental goals have to be met: 1) establishing trust and secure relationship between two smart cards, and 2) authentication, authorisation, and establishing trust in the current state of individual communicating applications.

We begin the discussion by providing the architecture of the CPAM. In section three, we describe the Platform Binding Protocol (PBP) that provides trust and establishment of secure binding between various platforms. Furthermore, in section four we discuss the Application Binding Protocol (ABP) that enables two applications to authenticate and ascertain trust in each other to accomplish application sharing. In section five, we briefly discuss structure of the application resource sharing request. Section six analyse the proposed protocols with respect to informal analysis that includes comparison with existing protocols, mechanical formal analysis using CasperFDR, and test implementation experience. Finally, in section seven we conclude the discussion and provide future research directions.

II. Cross-Platform Application Sharing Mechanism

In this section, we describe the CPAM initiative and then extend the discussion to the security and operational, goals and requirements for the proposed protocols that support the CPAM.

A. Framework for Cross-Platform Application Sharing

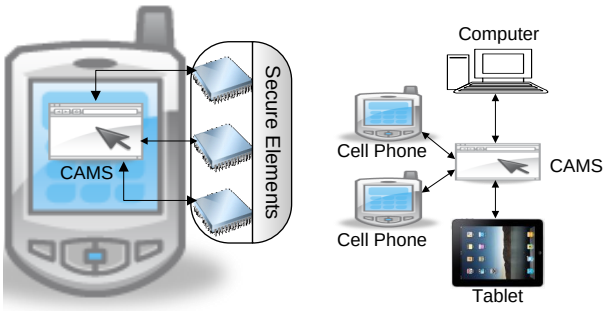
In the Cross-Platform Application Sharing (CPAS) architecture, a smart card acts like a node that is registered with a centralised system. The centralised system in our proposal is a software running on a host platform (e.g. computer, mobile phone, or tablets, etc.), which is referred as Card Application Management Software (CAMS) [6]. For a simplistic illustration, figure 1 shows two possibilities of the CPAS network.

In figure 1a, a mobile phone has three secure elements and each of them are connected to a CAMS hosted on the mobile phone. The CAMS can be hosted on an insecure platform and in the UCOM or the proposed CPAM, CAMS are not required to be secure. The only function of CAMS is to provide discoverability and interconnectivity to individual UCOM platforms. With discoverability, we mean that a platform registers itself

with the CAMS and thus it becomes discoverable to all other platforms in the network. The interconnectivity deals with the communication channel established between two (or more) secure elements. Therefore, figure 1a depicts the scenario in which multiple secure elements are connected to a host device (i.e. a mobile phone), and their interconnectivity and discoverability is handled by the CAMS installed on it.

On the other hand, figure 1b shows that host devices are connected with each other through their CAMS. Each individual host device may have multiple smart cards that are registered to their respective CAMS. Although, figure 1b depicts as if there is a single centralised CAMS. Which is incorrect, instead each host device has its own CAMS and there are no centralised CAMS. Thereby, if a particular device is not available, other platforms can still communicate with each other. Now in this scenario, a host device would discover other devices and register them. Here, we can divide this scenario into two possible cases. In first case, each individual host device advertises the connected smart cards to the entire network. Where in second case, each host device only provides the details of the respective CAMS and not the smart cards registered with it. For our proposed CPAM, we prefer the former as it provides a better privacy of individual smart cards.

Furthermore, the provision of whether an application supports CPAS is on the sole discretion of the respective SP. The UCOM architecture would provide two levels of application sharing: a) localised sharing, and b) CPAS network. The first option restricts the application sharing to the smart card on which the application is installed. Any client application that is not installed on it will not be able to access the shareable resources of the server application, and vice versa. This scenario is implemented in traditional smart card firewalls and supported by the UCOM [8, 16]. The second option allows application sharing with a client application, whether or not it is installed on the same platform.



(a) Cell Phone with Multiple Secure Elements

(b) Smart Cards attached to different Platforms

Figure 1. Cross-Platform Application Sharing Network

B. Trusted Environment & Execution Manager (TEM)

An important component of UCOM smart cards is the Trusted Environment & Execution Manager (TEM). The TEM is responsible for a) security assurance and validation of the platform, and b) user/TSM ownership management and delegation. The smart card runtime environment might conform to any of the smart card platforms or operating systems (e.g. Java Card or Multos).

A generic architecture of the TEM is illustrated in figure 2. For the sake of concision, we will only discuss those components of the TEM that are directly related to this paper.

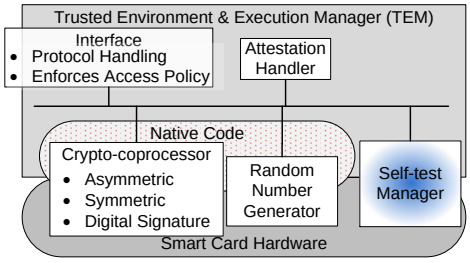


Figure 2. Generic Architecture of Trusted Execution & Environment Manager (TEM)

The two main components that are related to the proposed protocols are “Attestation Handler” and “Self-test Manager”. Both of these modules are used to provide a remote, dynamic, and ubiquitous security assurance and validation that the platform’s state is as it was at the time of a third party evaluation. This security assurance and validation mechanism is used during the application installation process to affirm to an SP that the smart card is as secure and reliable as it is stated by the evaluation certificate. An evaluation certificate is a cryptographically signed certificate issued by an evaluation body, and it is placed on the smart card by the respective card manufacturer [7].

The evaluation certificate will certify a unique signature key pair of the card manufacturer that it will use to issue certificates to the manufactured smart cards that conform to the evaluated product. During smart card manufacturing, the TEM will generate a signature key pair [17] that will be certified by the card manufacturer. On request for the platform assurance and validation (either from a cardholder, an SP, or an application), the TEM will evaluate the current state of the smart card. The result will be signed by the TEM and communicated to the requesting entity that can check whether the current state is the same as stated in the evaluation certificate. A point to note is that at present Common Criteria (CC) [18] or any other evaluation scheme, does not provide any such service but proposals presented in [7, 19] can be utilised. At the protocol layer, the assurance and validation requests are handled by the “Attestation Handler”.

The self-test mechanism implemented by the “Self-test Manager” validates both the hardware and software state of the smart card to be as it was at the time of evaluation. It is a two-part mechanism: tamper-evidence and reliability assurance. Smart cards are required to be a tamper-resistant device [20]. For this purpose smart card manufacturer implement hardware based tamper protections. The tamper-evidence process verifies whether the implemented tamper-resistant mechanisms are still in place and effective. The reliability assurance process verifies the software part of the smart card platform is not been tampered/modified.

For the tamper-evidence mechanism, one possible avenue is to utilise the Physically Unclonable Functions (PUFs) [21]

in conjecture with the TEM self-test implementations. In this paper, we consider them to provide the assurance that an SP is communicating with a hardware platform (i.e. to avoid a simulator attack [22]) and assure that all implemented security mechanisms (i.e. tamper-resistant mechanisms) are functioning properly. We are still analysing how the PUF or other tamper-evident mechanisms can be used in a secure and efficient way. However, for this paper we assume that such a mechanism is in place that can provide tamper-evidence to a requesting entity as a part of the proposed protocols. The software state of a platform can be validated by measuring its hash value. We can consider with reasonable assurance that the state of the smart card is as it was at the time of evaluation, if it matches with the one that is listed in the evaluation certificate.

Similarly, the TEM would also assist in ascertaining the current state of individual applications. When two SPs decide to have application sharing among their respective applications. Either both of the SPs can opt for a third party evaluation of their respective applications, or they can issue a certificate to each other's applications that contain the hash of application state. In both of these cases, the certified state of the application is treated as the trusted state by the other entity. During the CPAM protocols, individual TEMs will evaluate the current state of the respective application and communicate it to other applications, which will then verify whether the current stated is the trusted state.

The Platform Binding Protocol (PBP) will establish a secure binding between two devices (e.g. secure elements) facilitate the resource sharing between applications installed on them. The Application Binding Protocol (ABP) enables two applications to perform authentication, authorisation, and ascertain the trustworthiness in each other's state. On successful completion, it will generate a cryptographic key to bind the communicating applications.

C. Goals and Requirements for the Proposed Protocols

A CPAM based protocols for the UCOM architecture should meet the goals listed as below:

1) *Mutual Entity Authentication*: Entities authenticates to each other to avoid masquerading by a malicious entity.
2) Exchange of certified public keys between the entities to facilitate the key generation and entity authentication processes.
3) Mutual Key Agreement: Communicating parties will agree on the generation of a secret key during the protocol run.
4) Joint *Key Control*: Communicating parties will mutually control the key generation to avoid one party choosing weak keys.
5) *Key Freshness*: The generated key will be fresh to the protocol session.
6) Mutual *Key Confirmation*: Communicating parties will provide implicit/explicit confirmation that they have generated the same keys during a protocol run.
7) Known-Key Security: If a malicious user is able to obtain session key of a particular protocol run, it should not able her to retrieve the long-term secrets (*private keys*) or the *session keys* (future and past).
8) Unknown *Key* Share Resilience: If a malicious user retrieves the long-term key of an entity, it will enable her to impersonate the entity. Nevertheless, it should not facilitate her to impersonate other entities to it [23].
9) *Perfect Forward Secrecy*: If the long-term keys of communicating entities are compromised. An adversary should not be able to compromise previously generated keys.
10) Mutual *Non-Repudiation*: Communicating entities will not be able to deny that they have executed the protocol run with each other.
11) Trust Assurance (Trustworthiness): Communicating parties provide a validation proof that is dynamically generated during the protocol execution [7].
12) Privacy: A third party should not be able to know the identity of the communicating applications or devices.
13) Partial Chosen Key Attack: Protocols that claim to provide joint key control are susceptible to this attack [24]. In this attack, if two entities provide separate values to the key generation function then one entity has to communicate its contribution value to the other. The second entity can then compute the value of its contribution in such a way that it can dictate its strength (able to generate a partially weak key). However, this attack depends upon the computational capabilities of the second entity.
14) Avoid Simulator Attack: A malicious user masquerade a smart card on a computer (as a simulation). This can enable him to download an application on a simulated environment and perform reverse engineering on it, possibly revealing proprietary and sensitive data [22].

For formal definition of the italicised terms in the above list, readers are advised to refer to [17]. The notation used in the PBP and ABP protocols is listed in table I.

Table I
NOTATION USED TO DESCRIBE THE PROPOSED PROTOCOLS

S	Denotes a server application.
C	Denotes a client application.
SC^X	Denote a smart card platform X
SP^X	Denote the Service Provider of an application X.
X_i	Represents the identity of an entity X.
g^X	Current Diffie-Hellman exponential ($mod\ p$) generated by the entity X.
$Cert_X$	Signature key pair certificate of an entity X.
N_X	Random number generated by an entity X.
$A \rightarrow B$	Message sent by an entity A to an entity B.
$X\|\|Y$	Represents the concatenation of the data items X, Y in the given order.
$Sig_X(Z)$	Is the signature on data Z by an entity X using a signature algorithm [25].
$H(Z)$	Is the result of generating a hash of data Z.
$H_k(Z)$	Is the result of generating a keyed hash (HMAC) of data Z using key k.
$[M]^{K_e}_{K_m}$	Message M encrypted by the session encryption key K_e and then MAC is computed using the session MAC key K_m.
DH_{Group}	Details the Diffie-Hellman group that is used to generate the g^x [26].

III. PLATFORM BINDING PROTOCOL

The Platform Binding Protocol (PBP) is executed by two smart cards that are listed as SC^a and SC^b. Both smart cards can be part of the same CAMS or associated with

Table II
PLATFORM BINDING PROTOCOL DESCRIPTION

1.	$SC^a \rightarrow SC^b$	:	$SC^{a'}{}_i \| SC^{b'}{}_i \| N_{SC^a} \| H(N_{SC^a} \| g^{SC^a} \| SC^{b'}{}_i) \| DH_{Group}$
2.	$SC^b \rightarrow SC^a$	:	$SC^{b'}{}_i \| SC^{a'}{}_i \| N_{SC^b} \| H(N_{SC^b} \| g^{SC^b} \| SC^{a'}{}_i) \| DH_{GroupSel}$
3.	$SC^a \rightarrow SC^b$	:	$g^{SC^a} \| SC_i^{a'} \| SC_i^{b'} \| N_{SC^a} \| N_{SC^b}$
4.	$SC^b \rightarrow SC^a$	:	$g^{SC^b} \| N_{SC^b} \| [ReqVal \| SC_i^{a'} \| SC_i^b \| H(N_{SC^a} \| g^{SC^a} \| SC_i^{b'}) \| g^{SC^b} \| Cert_{SC^b} \| PEC]_{K_m}^{K_e}$
5.	$SC^a \rightarrow SC^b$	:	$[ReqVal \| Sign_{SC^a}(g^{SC^b} \| g^{SC^a} \| N_{SC^b} \| N_{SC^a} \| SC^a \| SC^b \| User_i \| H(SCOS)) \| Cert_{SC^a} \| PEC]_{K_m}^{K_e}$
6.	$SC^b \rightarrow SC^a$	:	$[Sign_{SC^b}(g^{SC^a} \| g^{SC^b} \| N_{SC^a} \| N_{SC^b} \| SC^a \| SC^b \| User_i \| H(SCOS)]_{K_m}^{K_e}$

two different CAMS. Both cases are accommodated by the protocol described in table II and discussed as below.

Message 1. The PBP is initiated by the smart card A referred as SC^a. The first message contains the pseudo identities of individual smart cards (e.g. $SC^{a'}$ and $SC^{b'}$), a random number generated by the SC^a. In addition, the SC^a will generate a Diffie-Hellman exponential g^{SC^a} but to avoid partial key chosen attack (see section III-C) it does not send the g^{SC^a}. Instead, it sends a commitment that is basically a hash generated on the g^{SC^a}, random number and recipient's pseudo identity.

Message 2. In response, the smart card B referred as SC^b will select a Diffie-Hellman group that it can support and include the selection as $DH_{GroupSel}$. The SC^b will generate a similar message as the SC^a (e.g. Message 1) and sends it to the SC^a including the $DH_{GroupSel}$. The Diffie-Hellman commitments are made from the both communicating entities and now in subsequent messages they can send the generated Diffie-Hellman exponential.

Message 3. The SC^a will send the Diffie-Hellman exponential to the SC^b along with pseudo-identities and random numbers generated by the communicating smart cards. On receipt, the SC^b will generate the Diffie-Hellman secret ($DH_{Secret} = (g^{SC^b})^{g^{SC^a}} \ mod\ n$. The session keys (i.e. encryption and Mac keys) are generated as: $K_e = H_{DH_{Secret}}(N_{SC^a} || N_{SC^b} || 1)$ and $K_m = H_{DH_{Secret}}(N_{SC^a} || N_{SC^b} || 2)$. The PBP master keys that will be used to generate session keys in all future communications are generated as: $K_E = H_{DH_{Secret}}(N_{SC^a} || N_{SC^b} || 0)$ and $K_M = H_{DH_{Secret}}(N_{SC^b} || N_{SC^a} || 0)$.

Message 4. In response, the SC^b will request for the platform assurance and validation proof (i.e. $ReqVal$) from the SC^a. Furthermore, the pseudo identity of the SC^a is appended with the true identity of the SC^b along with the commitment hash generated by the SC^a, Diffie-Hellman exponential and cryptographic certificate of the SC^b. Finally, the Product Evaluation Certificate (PEC) that is issued by third party evaluation that has tested the security and operational functionality of the smart card. The entire message except for the Diffie-Hellman Exponential and the generated random number is encrypted and Maced using the generated session keys (i.e. K_e and K_m).

On receipt of the message four, the SC^a will also generate the Diffie-Hellman secret along with session keys similar to the SC^b. It will then verify the SC^b's cryptographic certificate and PEC. If both smart cards are being evaluated by the same laboratory than this process will be simple as SC^a already trusts that particular evaluation laboratory. Otherwise, it will request the CAMS to traverse the certificate chain to find out whether the SC^a's evaluation laboratory is part of that certificate chain. Even if this fails, then the SC^a can request its respective card manufacturer that depending upon the provided certificate decides whether it should proceed with the binding or not. Therefore, only if SC^a successfully ascertain the validity of the certificate provider of the SC^b's certificate that it will proceed with the protocol.

Message 5. In response, the SC^a will proceed with a platform assurance and validation mechanism. On successful completion, the SC^a will generate a hash value of critical components and its referred as $H(SCOS)$ in the protocol description. The SC^a will append the Diffie-Hellman exponentials will generated random numbers and identities of communicating smart cards. The identity of current user of the SC^a is also concatenated with the message, which is then signed by the SC^a. The signed message is appended by the SC^a certificate and encrypted and Maced by the session keys.

The SC^b verifies the SC^a's signature and then validates whether the value of $H(SCOS)$ is same as it was at the time of evaluation, which is certified in the PEC of the SC^a. To verify the certificate chain, the SC^b will iteratively employ the similar procedure as SC^a discussed as part of the message four. Furthermore, the SC^b will also verify the identity of the user of the SC^a. The SC^b will record whether the user's identity is same for both smart cards or not. This information will be used by applications to decide whether they would like to establish a communication link with an application installed on a different user's smart card.

Message 6. The SC^b will initiate the platform assurance and validation mechanism, which generates the hash value of the critical components of the SC^b. It will append the Diffie-Hellman exponentials, random numbers and identities of communicating smart cards and current owner of the SC^b. Entire message is signed by the SC^b, which is encrypted and Maced by the session keys.

On receipt, the SC^a will verify the signature and validate the value of the $H(SCOS)$ to the same as the PEC specifics, which was sent to the SC^a in message four.

IV. APPLICATION BINDING PROTOCOL

The Application Binding Protocol (ABP) is executed between two applications that have an application sharing engagement. These two applications can be either on the same or different devices. The protocol listed in the table III accommodates the ABP when the respective applications are installed on two distinct devices. This protocol can also be used by applications even when they are installed on the same device. The protocol in table III is between a client, and server application represented as C and S, respectively.

Message 1. The protocol is initiated by the client application C, which is installed on a smart card SC^a. It will generate

Table III
APPLICATION BINDING PROTOCOL DESCRIPTION

1.	$C \rightarrow S$	:	$g^C \| \| N_C \| \| H_{S \rightarrow C}(C_i \| \| S_i \| \| SP^C{}_i \| \| SP_i^S \| \| g^C \| \| N_C) \| \| DH_{Group}$
2.	$S \rightarrow C$	:	$g^S \| \| N_S \| \| [ReqVal \| \| ReqUserID \| \| Sign_S(g^S \| \| g^C \| \| C_i \| \| S_i \| \| SP_i^C \| \| SP_i^C \| \| N_C \| \| N_S) \| \| Cert_S]_{K_m}^{K_e} \| \| DH_{GroupSel}$
3.	$C \rightarrow S$	:	$[ReqVal \| \| Sign_{SC^b}(H(S) \| \| SC_i^b \| \| UserID \| \| N_C \| \| N_S) \| \| Sign_C(g^S \| \| g^C \| \| SP_i^S \| \| SP_i^C \| \| N_C \| \| N_S) \| \| Cert_C \| \| Cert_{SC_C} \| \| PEC]_{K_m}^{K_e}$
4.	$S \rightarrow C$	:	$[Sign_{SC^a}(H(S) \| \| SC_i^a \| \| UserID \| \| N_C \| \| N_S) \| \| Cert_{SC_S} \| \| PEC \| \| ResLocator]_{K_m}^{K_e}$

a Diffie-Hellman exponential (g^C) and a random number. In addition, the application C also generates a keyed hash of identities of the client and server applications, and their respective SPs along with g^C and N_C. The rationale behind the generation of the keyed hash value is to avoid man-in-the-middle attack on the first message. To mount this attack, a malicious user has to gain knowledge of identities of individual applications and associated SPs and the secret key ($S \rightarrow C$). However, this message cannot avoid the replay attacks, which we deal with in subsequent messages. The message is then appended with the DH_{Group}.

On receipt of message one, the server application S installed on the smart card SC^b will verify the authentication credentials (i.e. keyed hash) and on a successful outcome, it will continue with the protocol. The application S will generate a Diffie-Hellman exponential and a random number. Subsequently, the application S will generate the session keys and long-term encryption and Mac keys in a similar manner as in message three of PBP (see section IV).

Message 2. The application S will generate a signature on the message containing the Diffie-Hellman exponential generated by both the client and server applications, identities of applications and their respective SP's along with generated random numbers. The signed message is appended by the server application's certificate along with requesting the client application's state validation and user authentication. The user authentication is an optional parameter in the ABP, which depends upon whether a server application allows application sharing with client applications that are issued to different users. If the server application allows application sharing with different user's client application then the parameter can be omitted. Otherwise, $ReqUserID$ will request the client application to provide the details of the registered owner of the smart card on which it resides. The message is then encrypted and Maced using the session keys.

On receipt of message two, the application C will first generate the DH_{Secret} and then session and long-term encryption and Mac keys. After this, it will proceed with verifying the Mac and decrypt the message two. First, it will validate the certificate $Cert_S$ and then verify the signature.

Message 3. The application C will then request the host smart card to validate its current state by generated the hash value. The smart card SC_b will generate the hash of the application and then sign it. The signed message includes hash of the application C, identities of the smart card and its owner (if requested by the server application in message two), random number generated by both applications. In addition, the client application also generates a signature on the message containing Diffie-Hellman exponentials generated by communicating applications along with identities of their respective SP's and generated random numbers. The signed message by the smart cards provides security assurance and validation that the client application, and the signed message by the client application provides entity authentication to the server application.

After the server application S receives the message three, it will first verify whether the generated hash of the application is same as certified by the either the SP of the application S or stated in PEC. If successful, then in will verify the signature generated by the application C. In case, the server application requests for the user's identity in message two then it will also verify whether the user identity provided by the message three is as required by the application S.

Message 4. In the final message of ABP, the server application S will request host smart card SC_a to generate the hash of the application S. The SC_a will generate the hash, append it with identities of the smart card and user (if required) along with generated random numbers. The resource locator referred as $ResLocator$ provides a handle to the shareable resources provided by the server applications. The $ResLocator$ uniquely identify the smart card on which the server application is installed, and the name of the resource that are being shared (e.g. SIO or RMI object, etc.).

On receipt, the application A will verify the state of the application S and if required to verify the identity of the current owner of the smart card SC_a on which the application S is installed.

V. CROSS-PLATFORM APPLICATION SHARING

At the time, when a client application requests the shareable resources, either in synchronous or asynchronous mode. It will use the long-term encryption and Mac keys generated during the ABP to calculate the session keys. The client application will generate the message illustrated as payload in figure 3. The message is appended with client and server application's identities. The smart card of the client application would then also generate session keys and encrypt the received message from client application. It appends the registered identities of the client and server application's smart cards. For brevity, we do not dive into the details of the session key generation, which can be left on the sole discretion of the communicating parties.

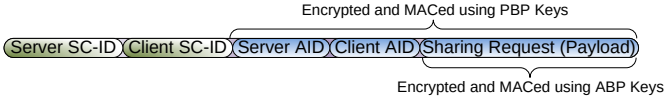


Figure 3. Cross-Platform Application Sharing Message

VI. ANALYSIS OF THE PROPOSED PROTOCOLS

In this section, we look at the proposed protocols in terms of informal analysis, mechanical formal analysis using

Table IV
PROTOCOL COMPARISON ON THE BASIS OF STATED GOALS (SEE SECTION III-C)

Goals	Protocols										
	STS	**AD**	**ASPeCT**	**JFK**	**T2LS**	**SCP10**	**SCP81**	**MM**	**SMM**	**PBP**	**ABP**
1	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
2	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
3	$*$	$*$	$*$	$*$	$*$					$*$	$*$
4	$*$	$*$	$*$	$*$						$*$	$*$
5	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
6	$*$		$*$	$*$		$*$	$*$	$*$	$*$	$*$	$*$
7	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
8	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
9	$*$		$*$	$*$						$*$	$*$
10	$(*)$	$(*)$	$*$	$*$	$*$	$+*$	$+*$	$+*$	$+*$	$*$	$*$
11				$*$						$*$	$*$
12										$*$	$*$
13	$*$		$*$	$*$						$*$	$*$
14										$*$	$*$

Note: $*$ means that the protocol meets the stated goal, $(*)$ represents that the protocol can modified to satisfy the requirement, and $+*$ illustrates that the protocol can meet the stated goal but require an additional pass or extra signature generation.

CasperFDR and finally providing the practical implementation and performance details.

A. Revisiting the Requirements and Goals

In this section, we take the security goals, and requirements stipulated in section III-C and provide a comparison of the proposed PBP and ABP protocols with the selected protocols like Station-to-Station (STS) [27], Aziz-Diffie (AD) [28] and Just Fast Key (JFK) [29]. Furthermore, we choose the ASPeCT protocol [30] that was created specifically for the mobile network environment, and the T2LS a trusted secure channel protocol [31].

From the smart card based environment, we have selected four protocols that are specifically designed for the smart card platform. These are the GlobalPlatform protocol SCP10 [32] and SCP81(based on SSL/TLS) [33], and Markantonakis-Mayes (MM) [34]. Finally, we included the Sirett-Mayes-Markantonakis (SMM) protocol [35] that is designed to securely download applications from an application server to a SIM card [20]; this protocol closely relates to the proposed protocols.

As shown in the table IV, the STS protocol meets the first 10 along with the goal 13. The remaining goals not met by the STS are because of the design architecture, and deployment environment, which did not require these goals. Similarly, the AD protocol does not meet certain goals. In the AD protocol, the user reveals her identity by sending the user certificate in clear along with the non-existence of key confirmation.

The most promising results were from the ASPeCt and JFK protocols that meet a large set of goals. Both of these protocols can be easily modified to provide the trust assurance (requiring additional signature). Furthermore, both of these protocols are vulnerable to the partial chosen key attacks. However, in the table IV we opt for the possibility that the JFK can be modified to meet this goal. The reason behind this is based on the entity that takes the initiator's role and in both protocols, a smart card takes the initiator role; thereby, avoiding partial chosen key attack.

The T2LS protocol meets the trust assurance goal by default; however, it does not meet most of the required goals of CPAS protocols (i.e. PBP, and ABP). A note in favour of the SCP10, SCP81, MM, and SM protocol is that they were designed with the assumption that an application provider has a prior trusted relationship with the smart card issuer; thus, implicitly trust the respective smart card. This assumption, which is nearly implausible to ascertain in the UCOM, is what makes these protocol not support a large number of the listed goals. Most of these protocols, to some extent have the similar architecture in which a server generates the key and then communicates that key to the client (i.e. read smart card). There is no non-repudiation as they do not use signatures in the protocol run.

Nonetheless, the proposed protocols PBP and ABP meet all the stated goals.

B. CasperFDR Analysis of the Protocols

The CasperFDR approach was adopted to test the soundness of the proposed protocol under the defined security properties. In this approach, the Casper compiler [39] takes a high-level description of the protocol, together with its security requirements. It then translates the description into the process algebra of Communicating Sequential Processes (CSP) [40]. The CSP description of the protocol can be machine-verified using the Failures-Divergence Refinement (FDR) model checker [41]. The intruder's capability modelled in the Casper script (Appendix A) for the proposed protocol is as below:

1) An intruder can masquerade any entity in the network.
2) It can read the messages transmitted by each entity in the network.
3) An intruder cannot influence the internal process of an agent in the network.

The security specification for which the CasperFDR evaluates the network is as shown below. The listed specifications are defined in the #Specification section of Appendix A:

1) The protocol run is fresh and both applications were alive.
2) The key generated by the SP and SC is not known to the intruder.
3) Entities have mutually authentication and key assurance at the conclusion of the protocol.

Table V
PROTOCOL PERFORMANCE MEASURES (MILLISECONDS)

Measures	SSL [36]	TLS [37]	Keberos [38]	PBP		ABP	
				Set One	Set Two	Set One	Set Two
Card Specification	32bit	32bit	32bit	16bit	16bit	16bit	16bit
Average Time	4200	4300	4240	4436.23	4628.35	2998.71	3091.38
Best Time	-	-	-	4078	4235	2906	3031
Worse Time	-	-	-	5469	5875	3922	4344
Standard Deviation	-	-	-	133.48	127.89	117.71	96.32

4) Long terms keys of communicating entities are not compromised.
5) User's identity is not revealed to the intruder.

The protocol description defined in the Casper script (Appendix A) is a simplified representation of the proposed protocol. The CasperFDR tool evaluated the protocol and did not find any feasible attack(s).

C. Practical Implementation

The proposed protocols does not provide any specific details of the cryptographic algorithms to be used during the protocol run. This choice is left to the respective SPs and smart cards. To provide a performance measure for the protocols, we have used Advance Encryption Standard (AES) [42] 128-bit key symmetric encryption with Cipher Block Chaining (CBC) [17] without padding for both encryption and MAC operations. The signature algorithm is based on the Rivest-Shamir-Aldeman (RSA) [17] 512-bit key. We have used SHA-256 [43] for the hash generation by the TEM. For Diffie-Hellman key generation, we used 2058-bit group with the 256-bit prime order subgroup specified in the RFC-5114 [26].

The architecture of the test-bed is based upon two entities, both implemented on smart cards. The protocols execute on 16-bit Java Cards, and the implementation takes 9799 bytes for PBP and 8374 bytes for ABP. The implemented protocols were executed for 1000 iterations and time taken to complete each iteration was recorded. The performance measures are taken from two different sets of 16-bit Java Cards, and an average of recorded measurements for both cards is listed in table V. For comparison, we have selected the performance of SSL [36], TLS [37], and Kerberos [38] on 32-bit smart cards. The performance measures of the SSL, TLS and Kerberos listed in the table V is with one node executing on a smart card and other on a PC. Therefore, unlike ABP and PBP only one communicating node is implemented on a smart card.

We consider that with adequate code optimisation we may be able to achieve better performance results. It should be noted that the performance measure is in reference to our implementation. Where the actual implementation will be required to take into account the SCOS and application size, and communication speed.

VII. CONCLUSION AND FUTURE RESEARCH DIRECTIONS

In this paper, we presented the CPAM architecture along with two protocols that support the proposal. The proposed protocols namely Platform Binding Protocol (PBP) and Application Binding Protocol (ABP), provide entity authentication, trusted assurance and validation, and key generation to devices and applications, respectively. Both protocols were compared with selected protocols as part of the informal analysis. We also provided the mechanism formal analysis based on the CasperFDR tool. Finally, the implementation details and performance measures of these protocols were listed.

The CPAS mechanism in specific and user centric security architecture for tamper-resistant devices in general is in its infancy. Therefore, as part of future research direction we would like to analyse the possible solutions to the platform assurance and validation mechanism. The security and reliability of the user centric security devices would heavily rely on the trustworthiness of this mechanism. Furthermore, how a network of user centric security devices can establish and maintain a dynamic trust matrix that calculates and assigns trust values to individual nodes. The conception of synchronous and asynchronous application sharing requires deliberations, and we aim for a well-defined and practically feasible system. Furthermore, the Java Card Shareable Interface Object (SIO) and Remote Method Invocation (RMI) methods are required to be studied and adequately modified for the CPAS. Finally, it would be beneficial to have PBP and ABP protocols based on elliptic curve cryptosystem, which will provide a comparison with the proposed approach.

VIII. ACKNOWLEDGEMENT

We would like to extend our appreciation to the anonymous reviewers for their valuable time and feedback. Additionally, thanks to Min Chen, Babar Mahmood, and Hisham Abbasi for patience while proof reading drafts.

REFERENCES

[1] *Multos: The Multos Specification,*, Online, . [Online]. Available: http://www.multos.com/
[2] *Java Card Platform Specification*, Online, Sun Microsystem Inc Specification Version 3.0.1, May 2009. [Online]. Available: http://java.sun.com/javacard/3.0.1/specs.jsp
[3] *ISO/IEC 18092: Near Field Communication - Interface and Protocol (NFCIP-1)*, International Organization for Standardization (ISO) Std., April 2004.
[4] (2011, February) GlobalPlatform Device: Secure Element Remote Application Management. Online. GlobalPlatform.
[5] G. Madlmayr, O. Dillinger, J. Langer, and J. Scharinger, "Management of Multiple Cards in NFC-Devices," in *CARDIS '08: Proceedings of the 8th IFIP WG 8.8/11.2 international conference on Smart Card Research and Advanced Applications*, G. Grimaud and F.-X. Standaert, Eds. Berlin, Heidelberg: Springer, 2008, pp. 149–161.
[6] R. N. Akram, K. Markantonakis, and K. Mayes, "A Paradigm Shift in Smart Card Ownership Model," in *Proceedings of the 2010 International Conference on Computational Science and Its Applications*, B. O. Apduhan and M. Gavrilova, Eds. Fukuoka, Japan: IEEE CS, March 2010, pp. 191–200.

[7] ——, "A Dynamic and Ubiquitous Smart Card Security Assurance and Validation Mechanism," in *25th IFIP International Information Security Conference (SEC 2010)*, ser. IFIP AICT, K. Rannenberg and V. Varadharajan, Eds. Brisbane, Australia: Springer, September 2010, pp. 161–172.

[8] ——, "Firewall Mechanism in a User Centric Smart Card Ownership Model," in *CARDIS 2010: Proceeding of the 9th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Application*, ser. LNCS, D. Gollmann, J.-L. Lanet, and J. Iguchi-Cartigny, Eds. Passau, Germany: Springer, April 2010, pp. 118–132.

[9] NFC Trials, Pilots, Tests and Live Services around the World. Online. NFC World. [Online]. Available: http://www.nearfieldcommunicationsworld.com/list-of-nfc-trials-pilots-tests-and-commercial-services-around-the-world/

[10] "StarBucks: Mobile Applications," Online, Visited May 2011. [Online]. Available: http://www.starbucks.com/coffeehouse/mobile-apps

[11] J. Guaus, L. Kanniainen, P. Koistinen, P. Laaksonen, K. Murphy, J. Remes, N. Taylor, and O. Welin, "Best Practice for Mobile Financial Services: Enrolment Business Model Analysis," Mobey Forum Mobile Financial Services Ltd., Helsinki, Finland, Online, June 2008. [Online]. Available: http://www.mobeyforum.org/files/bestpractice/Best%20Practices%20for%20MFS%20Enrolment%20Business%20model%20analysis%20final.pdf

[12] R. N. Akram, K. Markantonakis, and K. Mayes, "User Centric Security Model for Tamper-Resistant Devices," in *the 8th IEEE International Conference on e-Business Engineering (ICEBE 2011)*, J. Li and J.-Y. Chung, Eds. Beijing, China: IEEE Computer Science, October 2011.

[13] "National Strategy for Trusted Identities in Cyberspace," Department of Homeland Security, USA, Draft Proposal, June 2010. [Online]. Available: http://www.dhs.gov/xlibrary/assets/ns_tic.pdf

[14] "Future Networks and the Internet: Early Challenges Regarding the "Internet of Things"," Commission of the European Communities, Brussels, Commisision Staff Working Document SEC(2008) 2516, September 2008. [Online]. Available: http://ec.europa.eu/information_society/eeurope/i2010/docs/future_internet/swp_internet_things.pdf

[15] H. Kopetz, "Internet of Things," in *Real-Time Systems*, ser. Real-Time Systems Series. Springer US, 2011, pp. 307–323. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-8237-7_13

[16] R. N. Akram, K. Markantonakis, and K. Mayes, "Application-Binding Protocol in the User Centric Smart Card Ownership Model," in *the 16th Australasian Conference on Information Security and Privacy (ACISP)*, ser. LNCS, U. Parampalli and P. Hawkes, Eds. Melbourne, Australia: Springer, July 2011, pp. 208–225.

[17] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC, October 1996.

[18] *Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model, Part 2: Security functional requirements, Part 3: Security assurance requirements,*, Common Criteria Std. Version 3.1, August 2006.

[19] D. Sauveron and P. Dusart, "Which Trust Can Be Expected of the Common Criteria Certification at End-User Level?" *Future Generation Communication and Networking*, vol. 2, pp. 423–428, 2007.

[20] K. Mayes and K. Markantonakis, Eds., *Smart Cards, Tokens, Security and Applications*. Springer, 2008.

[21] H. Busch, M. Sotáková, S. Katzenbeisser, and R. Sion, "The PUF promise," in *Proceedings of the 3rd international conference on Trust and trustworthy computing*, ser. TRUST'10. Berlin, Heidelberg: Springer-Verlag, June 2010, pp. 290–297. [Online]. Available: http://portal.acm.org/citation.cfm?id=1875652.1875675

[22] R. N. Akram, K. Markantonakis, and K. Mayes, "Simulator Problem in User Centric Smart Card Ownership Model," in *6th IEEE/IFIP International Symposium on Trusted Computing and Communications (TrustCom-10)*, H. Y. Tang and X. Fu, Eds. HongKong, China: IEEE CS, Dec 2010.

[23] S. Blake-Wilson, D. Johnson, and A. Menezes, "Key Agreement Protocols and Their Security Analysis," in *Proceedings of the 6th IMA International Conference on Cryptography and Coding*. London, UK: Springer-Verlag, 1997, pp. 30–45.

[24] C. Mitchell, M. Ward, and P. Wilson, "Key Control in Key Agreement Protocols," *Electronics Letters*, vol. 34, no. 10, pp. 980 –981, May 1998.

[25] C. Furlani, *FIPS 186-3 : Digital Signature Standard (DSS)*, Online, National Institute of Standards and Technology (NIST) Std., June 2009.

[26] M. Lepinski and S. Kent, "RFC 5114 - Additional Diffie-Hellman Groups for Use with IETF Standards," , Jan 2008.

[27] W. Diffie, P. C. Van Oorschot, and M. J. Wiener, "Authentication and Authenticated Key Exchanges," *Des. Codes Cryptography*, vol. 2, pp. 107–125, June 1992.

[28] A. Aziz and W. Diffie, "Privacy And Authentication For Wireless Local Area Networks," *IEEE Personal Communications*, vol. 1, pp. 25–31, First Quarter 1994.

[29] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, and O. Reingold, "Just Fast Keying: Key Agreement in a Hostile Internet," *ACM Trans. Inf. Syst. Secur.*, vol. 7, May 2004.

[30] G. Horn, K. M. Martin, and C. J. Mitchell, "Authentication Protocols for Mobile Network Environment Value-Added Services," in *IEEE Transactions on Vehicular Technology*, vol. 51. IEEE, Mar 2002.

[31] Y. Gasmi, A.-R. Sadeghi, P. Stewin, M. Unger, and N. Asokan, "Beyond Secure Channels," in *the 2007 ACM workshop on Scalable Trusted Computing*. NY, USA: ACM, 2007, pp. 30–40.

[32] *GlobalPlatform: GlobalPlatform Card Specification, Version 2.2,*, GlobalPlatform Std., March 2006.

[33] *Remote Application Management over HTTP, Card Specification v 2.2 - Amendment B*, Online, GlobalPlatform Specification, September 2006.

[34] K. Markantonakis and K. Mayes, "A Secure Channel Protocol for Multi-application Smart Cards based on Public Key Cryptography," in *CMS 2004 - Eight IFIP TC-6-11 Conference on Communications and Multimedia Security*, D. Chadwick and B. Prennel, Eds. Springer, September 2004, pp. 79–96.

[35] W. G. Sirett, J. A. MacDonald, K. Mayes, and C. Markantonakis, "Design, Installation and Execution of a Security Agent for Mobile Stations," in *Smart Card Research and Advanced Applications, 7th IFIP WG 8.8/11.2 International Conference, CARDIS*, ser. LNCS, J. Domingo-Ferrer, J. Posegga, and D. Schreckling, Eds., vol. 3928. Tarragona, Spain: Springer, April 2006, pp. 1–15.

[36] P. Urien, "Collaboration of SSL Smart Cards within the WEB2 Landscape," *Collaborative Technologies and Systems, International Symposium on*, vol. 0, pp. 187–194, 2009.

[37] P. Urien and S. Elrharbi, "Tandem Smart Cards: Enforcing Trust for TLS-Based Network Services," *Applications and Services in Wireless Networks, International Workshop on*, vol. 0, pp. 96–104, 2008.

[38] A. Harbitter and D. A. Menascé, "The Performance of Public Key-Enabled Kerberos Authentication in Mobile Computing Applications," pp. 78–85, 2001.

[39] *Casper: A Compiler for the Analysis of Security Protocols*, ser. Journal of Computer Security, June 1998.

[40] C. A. R. Hoare, *Communicating Sequential Processes*. New York, NY, USA: ACM, 1978, vol. 21, no. 8.

[41] P. Ryan and S. Schneider, *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley Professional, 2000.

[42] Joan Daemen and Vincent Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Berlin, Heidelberg, New York: Springer Verlag, 2002.

[43] *FIPS 180-2: Secure Hash Standard (SHS)*, National Institute of Standards and Technology Std., 2002.

APPENDIX

A. CasperFDR Script for the Platform Binding Protocol

```
#Free variables
datatype Field = Gen|Exp(Field, Num) unwinding
2
halfkeySCA, iMsg, rMsg, halfkeySCB, EnMaKey :
Field
SCB, SCA, User: Agent
gSCB, gSCA: Num
nSCB, nSCA, SCOSAHash, SCOSBHash: Nonce
VKey: Agent->PublicKey
SKey: Agent->SecretKey
h: HashFunction
InverseKeys = (VKey, SKey), (EnMaKey, \
EnMaKey), (Exp, Exp), (Gen, Gen)
----------------------------------------
#Protocol description
0.    -> SCA : SCB
[SCB!=SCA]
<iMsg:=Exp(Gen,gSCA)>
1. SCA -> SCB : SCA, nSCA, H(iMsg)%hashSCA
<rMsg:=Exp(Gen,gSCB)>
2. SCB -> SCA : SCB, nSCB, H(rMsg)%hashSCB
3. SCA -> SCB: SCA, SCB, nSCA, nSCB,
iMsg%halfkeySCA,
```

```
<EnMaKey := Exp(halfkeySCA, gSCB)
[hashSCA==h(halfkeySCA)]
4. SCB -> SCA : SCA, SCB, rMsg%halfkeySCB,
{SCB, nSCA, nSCB}{EnMaKey}
<EnMaKey := Exp(halfkeySCB, gSCA)>
[hashSCB==h(halfkeySCB)]
5. SCA -> SCB : {{gSCA, gSCB, User,
SCOSAHash}{SKey(SCA)}}{EnMaKey}
6. SCB -> SCA : {{gSCA, gSCB, User,
SCOSBHash}{SKey(SCB)}}{EnMaKey}
----------------------------------------
#Actual variables
SCardOne, SCardTwo, USER, MaliciousEntity:
Agent
GSCB, GSCA, GMalicious: Num
NSCB, NSCA, NMalicious: Nonce
----------------------------------------
#Processes
INITIATOR(SCA,SCB, User, gSCA, nSCA,
SCOSAHash)knows SKey(SCA), VKey
RESPONDER(SCB,SCA, User, gSC, nSCB, SCOSBHash)
knows SKey(User), SKey(SC), VKey
----------------------------------------
#System
INITIATOR(SCardA, SCardB, GSCA, NSCA,
SCOSAHASH)
RESPONDER(SCardB, SCardA, USER, GSCB, NSCB,
SCOSBHASH)
----------------------------------------
#Functions
symbolic VKey, SKey
----------------------------------------
#Intruder Information
Intruder = MaliciousEntity
IntruderKnowledge = {SCardA, SCardB,
MaliciousEntity, GMalicious, NMalicious,
SKey(MaliciousEntity), VKey}
----------------------------------------
#Specification
Aliveness(SCA, SCB)
Aliveness(SCB, SCA)
Secret(SCA, EnMaKey, [SCB])
Secret(SCB, EnMaKey, [SCA])
Secret(SCA, User, [SCB])
Secret(SCB, User, [SCA])
Agreement(SCA, SCB, [EnMaKey])
Agreement(SCB, SCA, [EnMaKey])
----------------------------------------
#Equivalences
forall x, y : Num . Exp(Exp(Gen, x), y) =
Exp(Exp(Gen, y), x)
```

B. CasperFDR Script for the Application Binding Protocol

```
#Free variables
datatype Field = Gen|Exp(Field, Num) unwinding
2
halfkeyC, iMsg, rMsg, halfkeyS, EnMaKey :
Field
S, C, User: Agent
gS, gC: Num
nS, nC, AppS, AppC: Nonce
VKey: Agent->PublicKey
SKey: Agent->SecretKey
h: HashFunction
InverseKeys = (VKey, SKey), (EnMaKey,
EnMaKey), (Exp, Exp), (Gen, Gen)
----------------------------------------
#Protocol description
0.    -> C : S
[S!=C]
<iMsg:=Exp(Gen,gC)>
1. C -> S : C, nC, iMsg%halfkeyC
<EnMaKey := Exp(halfkeyC, gS);\
 rMsg:=Exp(Gen,gS)>
2. S -> C : S, nS, rMsg%halfkeyS
<EnMaKey := Exp(halfkeyS, gC)>
3. C -> S: nC, nS
4. S -> C : {{halfkeyC, halfkeyS, User, nC,
nS, h(AppS)}{SKey(S)}}{EnMaKey}
[rMsg==halfkeyS]
5. C -> S : {{halfkeyC, halfkeyS, User, nC,
nS, h(AppC)}{SKey(C)}}{EnMaKey}
[iMsg==halfkeyC]
----------------------------------------
#Actual variables
Server, Client, USER, MaliciousEntity: Agent
GS, GC, GMalicious: Num
NS, NC, APPS, APPC, NMalicious: Nonce
----------------------------------------
#Processes
INITIATOR(C,S, User, gC, nC, AppC)knows
SKey(C), VKey
RESPONDER(S,C, User, gS, nS, AppS) knows
SKey(S), VKey
----------------------------------------
#System
INITIATOR(Client, Server, GC, NC, APPC)
RESPONDER(Server, Client, USER, GS, NS, APPS)
----------------------------------------
#Functions
symbolic VKey, SKey
----------------------------------------
#Intruder Information
Intruder = MaliciousEntity
IntruderKnowledge = {Server, Client, APPC,
APPS, MaliciousEntity, GMalicious, NMalicious,
SKey(MaliciousEntity), VKey}
----------------------------------------
#Specification
Aliveness(C, S)
Aliveness(S, C)
Secret(C, EnMaKey, [S])
Secret(S, EnMaKey, [C])
Secret(S, User, [C])
Secret(C, User, [S])
Agreement(C, S, [EnMaKey])
Agreement(S, C, [EnMaKey])
----------------------------------------
#Equivalences
forall x, y : Num . Exp(Exp(Gen, x), y) =
Exp(Exp(Gen, y), x)
```